# INTRODUCCION A
# ALGORITMOS Y PROGRAMACIÓN
# EN TURBO PASCAL

ALFONSO PIO AGUDELO SALAZAR
**Instructor Asociado**

**Requisito para promoción a:**
**Profesor Asistente**

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE MANIZALES
1995

## TABLA DE CONTENIDO

# INTRODUCCIÓN

Todos los seres humanos resolvemos problemas a diario, hay diferentes maneras de hacerlo, sinembargo no siempre esos problemas se resuelven con eficiencia y eficacia, por lo que se hace necesario comprender ciertos conceptos para lograr dar la mejor resolución a dichos problemas y específicamente con la ayuda del computador. En el presente texto, se abordará en primera instancia el tema sobre algoritmos, de una manera general para luego entrar a estudiar el lenguaje de programación Turbo Pascal, en la resolución de problemas a través del computador.

Los lenguajes permiten comunicarse con el computador, es un medio para decirle a la máquina cómo debe hacer una tarea.

Un lenguaje es un conjunto de representaciones o manifestaciones por medio de las cuales puede existir comunicación entre diferentes entidades. Los animales en algunas ocasiones manejan su propio lenguaje, el hombre y el perro pueden manejar un lenguaje a través de silbidos y ciertas palabras que logra entender la fiera y que al escucharlas realiza alguna pirueta, o simplemente realiza un trabajo; el ser humano utiliza varias formas de lenguaje para comunicarse no sólo con sus congéneres (lenguaje escrito u oral), sino también con las máquinas, a éstos últimos se les denomina lenguajes de programación con computadoras.

Todos los lenguajes tienen formas muy similares de estructuras o sentencias. Al igual que en el lenguaje escrito de los seres humanos, existen en los lenguajes de programación reglas claramente definidas de **gramática**: morfología (conocimiento de los códigos), sintaxis ( forma de escribir código para formar sentencias), ortografía

(cómo escribir código correctamente); así como de **semántica** (sentido o entendimiento de lo que se escribe a través del código).

Existen muchos lenguajes de programación, a medida que sube de generación el lenguaje es más amigable, más sencillo de manejar y se vuelve más ágil la programación:

Lenguajes de bajo nivel: son los que más se acercan al lenguaje de máquina, como lo es lenguaje ensamblador.

Lenguaje de alto nivel: Son más amigables para el usuario, pero obviamente, para que la máquina ejecute instrucciones de estos lenguajes se necesita un compilador o un interpretador de programas. Estos se pueden clasificar:

Lenguajes de segunda generación: Fortran, Basic, Algol.

Lenguajes de tercera generación: Pascal, C, Etc.

Lenguajes de cuarta generación ( 4GL) : SQL,QBE,QUEL.

Lenguajes orientando a objetos: C++, Turbo Pascal 7.0, Scheem.

# INTRODUCCION A
# ALGORITMOS Y PROGRAMACIÓN
# EN TURBO PASCAL

# 1. ALGORITMO

La palabra algoritmo proviene del nombre de un matemático árabe llamado Mohammed Alkhowarizmi, personaje que existió en el siglo IX y a quien se le atribuyen los enunciados de las operaciones aritméticas básicas. Los algoritmos permiten describir los pasos para la realización de una tarea; en realidad es una forma de describir la solución a un problema paso a paso; es un medio por el cual se explica cómo puede resolverse un problema, utilizando necesariamente una aproximación paso a paso. Puede formularse de muchas maneras siempre y cuando no se realice de forma ambigua, debe ser preciso, indicar el orden de realización de cada paso, debe indicar también claramente cuando dejar de hacer algo y cuando continuar haciendo algo más, debe describir cómo elegir entre una o varias alternativas. Si se hace el seguimiento a un algoritmo más de una vez, debe obtenerse siempre el mismo resultado. Un algoritmo debe ser finito: en algún momento debe terminar.

Ejemplo: Algoritmo para el arranque de un vehículo

1. Verificar si el vehículo tiene activada la alarma
2. Si la alarma se encuentra activada
3. Desactivar alarma
4. Abrir la puerta para permitir el ingreso del conductor
5. **Encender**

Algoritmo para **encender** un vehículo

1. Si la palanca de cambios se encuentra engranada (posición diferente de neutro)
2. Accionar freno
3. Accionar embrague
4. Desengranar
5. Introducir llave de encendido
6. Girar llave de encendido y acelerar
7. Si hay encendido
8. Soltar embrague y acelerar

Podemos definir un algoritmo como una serie de acciones lógicas bien determinadas, en número finito de ellas las cuales se deben realizar en un orden tal que permitan obtener la solución a un problema. Otro ejemplo:

1. Levantarse de la cama.
2. Dirigirse al baño y bañarse.
3. Afeitarse.
4. Vestirse
5. Si hay tiempo
6. Desayunar
7. Salir a tomar el bus.
8. Si se demora y se hace tarde.
9. abordar un taxi.
10. Si no, abordar el bus

El número de pasos que tiene el algoritmo, tiene una correlación directa con el número de instrucciones que haya que realizar en la solución de un problema sin importar el lenguaje que se utilice.

Para llegar al algoritmo, es necesario analizar el proceso y luego convertirlo en una serie de pasos que deben realizarse de alguna manera y que por último se codifica en cualquier lenguaje de programación.

Algoritmo que maneja las transacciones de un cajero automático.

1. Introducir la tarjeta.
2. Teclear la clave.
3. Verificar la clave
4. Si la clave es correcta.
5. Repita seleccionar operación
6. Si la operación es mostrar saldo.
7. Muestra el saldo.
8. Si la operación es retirar.
9. Solicite la suma a retirar
10. Descuente la suma del saldo.
11. Entregue la plata.
12. Si la operación es consignar.
13. Solicite la suma a ingresar.
14. Solicite la entrada del sobre.
15. Sume el valor al saldo.
16. Hasta que la operación sea terminar

## 1.1. Formalización de los Algoritmos

Pasar de un algoritmo a una secuencia de instrucciones de un lenguaje determinado, es a veces un poco complicado. Para ello muchos autores se han inventado una forma de lenguaje muy parecido a los lenguajes de alto nivel, el pseudocódigo, que formaliza los algoritmos acercándolos más a las instrucciones de cualquier lenguaje de programación, haciendo aún más sencillo el paso de uno al otro.

El esfuerzo mayor se centra en buscar cuáles son las variables, cómo se formalizan las condiciones y por último como se debe escribir una instrucción.

**Se recomienda el Libro: Algoritmos Estructurados. Tamayo A. , Alonso. Universidad Nacional de Colombia, Sede Manizales. (De venta en Publicaciones de La Universidad Nacional)**

## 2. PROGRAMACIÓN EN TURBO PASCAL Versión 7.0

Un lenguaje de programación es todo un conjunto de reglas que permiten expresar en forma escrita un algoritmo, para que la computadora pueda ejecutarlo; el lenguaje de programación permite la comunicación con la máquina para la solución de problemas.

En un programa se materializa una forma de solución del problema para que el computador pueda resolverlo. El programa, escrito en algún lenguaje de programación debe tener una sintaxis, es decir unas reglas de cómo escribir código de programas; y una semántica, es decir una interpretación o significado de lo que hace el programa.

Entre los pioneros en el desarrollo de una metodología de programación se encuentra el Profesor Niklaus Wirth, quien es uno de los desarrolladores del método de refinamiento a pasos y creador del lenguaje Pascal. Esta metodología es más ampliamente precisada y fundamentada por Dijkstra y Gries quienes proponen una técnica en el desarrollo de programas "guiado por objetivos" que consiste en una construcción simultánea del programa y la prueba de su corrección. Son tres métodos de solución de problemas implicados en la concepción de Dijkstra: Dividir y conquistar, Reducción de Casos e iteración. Desde otro punto de vista, pero con una gran coherencia se encuentra Jackson quien muestra las ventajas obtenidas al hacer corresponder la estructura de un programa con la de sus datos.

El lenguaje Pascal es un lenguaje de alto nivel, de propósito general, fue diseñado por el Profesor Niklaus Wirth de la Universidad de Zurich por el año 1970. Encontramos en el mercado una gran cantidad de versiones del lenguaje Pascal, no sólo del Pascal

estándar, sino también del Turbo Pascal, siendo una de las más recientes la versión 7.0, la cual se utilizará en el presente texto para desarrollo de los ejemplos.

En la elaboración de un programa, se debe hacer abstracción sobre los objetos o entidades del mundo REAL que intervienen en el proceso y representarlos con algún tipo de dato o estructura de datos en el programa, por ejemplo se debe asociar algo del mundo REAL (edad de una persona) con el nombre de una variable (EDAD) dentro del programa.

## 2.1. Forma de un programa en Turbo Pascal

**Cabecera**
**Directivas**
**Declaraciones y Definiciones**
- **Constantes**
  - **Tipificadas**
  - **No tipificadas**
- **Tipos**
- **Variables**
- **Etiquetas**

**Sección de código**

Un programa es una secuencia de instrucciones; estas representan los pasos que se deben seguir para elaborar una tarea, es decir un programa consiste en un código tal que le permita a la computadora realizar la tarea prescrita en un algoritmo.

## 2.2. Tipos de datos estándar en Turbo Pascal

**Byte:** ocupa un byte = 8 bits, es un valor numérico sin signo que puede variar desde 0 hasta 255, puede ser sustituida por un entero con una longitud siempre que su valor no exceda el rango del byte, o sea que el máximo entero que almacena será:
$2^8-1=255$.

**Word**: palabra, ocupa dos bytes en memoria, su dominio es de 0 a 65535.

**Shortint:** son los enteros cortos, ocupan un byte de memoria su dominio está entre -128 a 127.

**Integer:** ocupa dos bytes = 16 bits, es un valor numérico que pertenece a los enteros con signo que puede variar desde $-2^{15}=-32768$ hasta $2^{15}-1=32767$.

**Longint:** llamados enteros largos y ocupan 4 bytes de memoria para su almacenamiento, su rango es: $-2^{31}=-2147483648$ a $2^{31}-1=2147483647$.

**Real:** ocupa seis bytes = 48 bits, es un valor numérico que contiene punto decimal y puede variar desde -10E38 a -10E-38, y en números positivos 10E38 a 10E-38.

**Char:** ocupa un byte = 8 bits, puede almacenar cualquier caracter y no puede ser usado en operaciones aritméticas. Se usa para la manipulación y comparación de textos.

**String:** ocupa un byte más que su longitud definida, puede almacenar cualquier cadena de caracteres de igual longitud a la longitud definida, puede tener una longitud máxima de 255 caracteres.

**Boolean:** ocupa un byte, se usa para almacenar valores lógicos: verdadero (TRUE) o falso (FALSE).

## 2.3. Tipos de datos definidos por el usuario en Turbo Pascal

**Escalar:** Consiste en una lista de los valores que pueden ser tomados por un variable de ese mismo tipo necesita un Byte de memoria y puede tener hasta 256 elementos, ejemplo:

```
TYPE
    Profesion= (abogado, contador, ingeniero);
VAR
    empleado: profesion;
```

La variable empreado es de tipo **profesion** y puede tener cuaquiera de los tres valores indicados en la declaración de tipo: **abogado, contador, ingeniero**; y ningún otro valor.

**Subintervalo o Subrango:** Este tipo se define por dos constantes, las cuales deben ser distintas y del mismo tipo. El tipo de las constantes puede ser de tipo entero, caracter o escalar. Los subrangos de reales no son permitidos.

Ejemplo:

```
TYPE
    Caracminus='a'..'z';
    Digitos='0'..'9'
```

```
VAR
        Letraminus: Caracminus;
        Numeros:Digitos;
```

**Registro:** es una agrupación de otros tipos de datos en un nuevo tipo de dato. Su requerimiento de memoria es igual a la sumatoria de la memoria gastada por todos los campos definidos en el registro.

Ejemplo:

```
VAR
        alumno= RECORD
                        Nombre: STRING[12];
                        Apellido: STRING[13];
                        nota1: REAL;
                        sexo: byte;
                END;
```

Cuando se usan registros en una estructura de datos, hay ventaja puesto que todos los campos están conectados lógicamente entre sí, formando una sola estructura.

**Arreglo**: es una estructura que nos permite tener una agrupación de datos toda del mismo tipo. Su requerimiento de memoria será el producto de la longitud del arreglo por la memoria que ocupa el tipo de dato del que está formado. Cualquier tipo de dato, puede ser incluido en un arreglo, excepto el tipo File.

Ejemplo:

```
TYPE
        alumno= RECORD
            Nombre: STRING[12];
            Apellido: STRING[13];
            nota1: REAL;
            sexo: BYTE;
         END;
VAR
        curso: ARRAY [1..25] OF alumno;
```

La anterior estructura permite representar una estructura de datos para un grupo de 25 alumnos.

**Archivos de texto:** se conforman de líneas que terminan en un retorno de carro o avance de línea que es el delimitador. La cantidad de memoria en disco que ocupa uno de estos archivos es igual en bytes al número de caracteres ASCII que éste ocupa. Una declaración de archivo de texto es:

```
VAR
    archivo: TEXT;
```

En este caso el identificador de variable es **archivo**. La palabra reservada TEXT en Pascal indica que se trata de un archivo de texto.

**Archivos tipeados:** se conforman de unidades o elementos de datos de algún tipo; pueden ser estos tipos estándar o definidos por el usuario.
Una declaración de un archivo tipeado con tipos estándar sería:

```
VAR
    matriz: FILE OF REAL;
```

Una declaración de un archivo tipeado con registros definidos por el usuario sería:

```
TYPE
        alumno= RECORD
              Nombre: STRING[12];
              Apellido: STRING[13];
              nota1: REAL;
              sexo: BYTE;
         END;
VAR
    curso: FILE OF alumno;
```

## 2.4. Declaraciones y Definiciones

En este bloque se declaran y definen etiquetas, constantes, tipos, variables, procedimiento y funciones.

Las constantes se definen de la siguiente manera:

```
CONST
    beta=4.23443;
```

Una definición de tipo es por ejemplo:

```
TYPE
    nombre= STRING[15];
```

Las declaraciones de variables, tienen la siguiente forma:

```
VAR
    saldo,ingreso,retiro: REAL;
    mes,dia: INTEGER;
```

Una declaración de procedimiento sería:

```
PROCEDURE calculo_de_raices;
BEGIN
        instrucciones para el cálculo de raíces;
END;
```

Ejemplo en la declaración de función:

```
FUNCTION potencia(base:REAL; exponente: INTEGER): REAL;
VAR i:INTEGER;
BEGIN
        res:=0;
        FOR i:=1 TO exponente DO
                res:=res*base;
        potencia:=res;
END;
```

## 2.5. Expresiones

Una expresión puede ser una fórmula algebraica o un conjunto proposiciones lógicas o booleanas que conducen a un valor lógico o booleano; en las expresiones intervienen dos tipos de elementos: los **operadores** y los **operandos**.

Entre los **operadores** se encuentran: los aritméticos tales como suma, resta, multiplicación y división, las expresiones con operadores aritméticos conducen a resultados numéricos; y los operadores de tipo lógico o booleano como **AND, OR, NOT** conducen a un valor lógico o booleano (**falso** = **FALSE** o **verdadero**= **TRUE**). También se pueden realizar operaciones sobre cadenas de caracteres, las cuales producen cadenas de caracteres, tal es el caso de la suma de cadenas de caracteres.

Las expresiones matemáticas se pueden ver de la siguiente manera:

$x := 3 * 2 + 4 + 10;$

$y := 5*2+4*3-3 * 6;$

$z := 2 / 3 * (x + y);$

Se observa que en las expresiones puede haber una confusión en el sentido de saber cuál es el valor que se le asigna a la variable debido a la secuencia de los operandos, para evitar esto se necesita saber el orden de prioridad de las operaciones: primero se hacen las operaciones que se encuentran dentro de paréntesis, luego los productos y divisiones y por último se hacen las sumas y restas.

Para las operaciones anteriores, los resultados serán:

a $x$ se le asignará el valor 20;

a $y$ se le asignará el valor 4;

a $z$ se le asignará el valor 16.

Ejemplos de expresiones booleanas:

Dado que los valores booleanos de: ***x*** = TRUE, ***y***=FALSE;

***x*** **AND** ***y*** valdrá **FALSE**

***x*** **OR** ***y*** valdrá **TRUE**

***x*** **AND NOT** ***y*** valdrá **TRUE**

Los operandos son aquellas instancias (variables, constantes) dentro de una expresión sobre los cuales se realizan las operaciones.

## 2.6. Entradas y salidas

Las entradas son valores que el programa necesita para su ejecución. Cuando el sistema digestivo va a funcionar es necesario que en él entre un alimento. De igual manera hay programas que para comenzar o continuar su funcionamiento es necesario que se le dé como entrada uno o varios valores.

En el caso de un programa que sume dos números, se necesita entrarle los valores de los números a sumar, para que dé como salida la suma. La sentencia **READ** es típica instrucción de entrada de datos, lee del teclado los valores de variables, lee datos de un archivo de disco, o de cualquier otro dispositivo de entrada.

Las sentencias de salida se especifican dentro del programa y muestran a través de algún dispositivo de salida las respuestas del problema. La sentencia **WRITE** es una típica instrucción de salida de datos, puede mostrar datos sobre la pantalla, por la impresora, a un archivo sobre un disco, o a otro dispositivo de salida.

En el ejemplo de un cajero inicialmente una entrada será cuando el cajero le solicite al cliente la tarjeta y el usuario entra la clave de la persona, si ésta es correcta tendrá una salida en la que le mostrará al cliente las operaciones que podrá realizar el cajero, después habrá otra entrada cuando el cliente selecciona la operación y de acuerdo a lo seleccionado tendrá una salida que informará el saldo, o entradas de los valores a retirar y consignar.

**Algoritmo para empezar a editar programas en turbo pascal:**

**Nota:** se supone que el usuario al momento, debe tener alguna destreza en el manejo tanto del sistema operativo DOS, así como en el manejo de la máquina (teclado, ratón, etc), y en especial el manejo del editor de Turbo Pascal 7.0.

1. Cargar el DOS. (Sistema Operacional).
2. Ubicar el directorio BP, en el dispositivo de almacenamiento. >**cd bp** <ENTER>
3. Ubicar el subdirectorio BIN. >**cd bin** <ENTER>
3. Ejecutar la orden>**turbo** <ENTER>.

Inmediatamente Turbo Pascal se sitúa en el editor. El editor de Turbo Pascal consiste en un programa por medio del cual se escriben y/o modifican programas. Si el nombre de un archivo se encuentra en el mecanismo de almacenamiento, con el editor Turbo pascal se podrá modificar el archivo.

Un ejemplo muy simple:

```
PROGRAM caso1;
BEGIN
      WRITE('UNIVERSIDAD');
      READLN;
END.
```

La palabra **PROGRAM** es reservada del lenguaje; siempre debe ir al inicio del programa, caso1 es una palabra dada por el usuario e indica el nombre del programa, el signo punto y coma (;) hace parte de la sintaxis, por lo tanto **no** puede ser ignorado, este signo (;) siempre se utilizará en la separación de instrucciones y debe ir al final de cada sentencia. El siguiente renglón contiene una palabra reservada **BEGIN** que significa comienzo del programa. La tercera línea contiene una sentencia o instrucción de salida **WRITE** que significa **escribir** (en este caso sobre la pantalla), lo que se encuentra entre paréntesis (argumentos); en este caso se trata de un texto que deseamos que aparezca por pantalla, debe ir entre comillas (la tecla de comilla con que abre es la

misma con que cierra). El punto y coma al final de la tercera línea, recordemos que la sentencia ha terminado. La cuarta línea posee una instrucción **READLN**, que permite hacer una espera para lograr que la salida por pantalla se vislumbre hasta que el usuario pulse una entrada. Por último, la quinta línea posee otra palabra reservada **END**, que significa fin del programa, además sigue el punto (.) que significa punto final para el programa.

**Compilación y ejecución de un programa**: Asociado a cualquier lenguaje de programación encontramos un programa (por lo general realizado en un lenguaje de bajo nivel llamado **compilador**), cuya función es traducir los programas escritos en dicho lenguaje (**código o programa fuente**), al lenguaje de **máquina (código o programa objeto**) que en últimas es el único lenguaje que puede manejar la unidad central de proceso en el computador. Todo **programa fuente** debe ser traducido a un **programa objeto** antes de ser ejecutado por la máquina, a éste proceso se le denomina **compilación**.

Al correr o ejecutar un programa en Turbo Pascal, primero se debe pulsar la tecla **F10** para activar del menú principal, se selecciona la opción **RUN** pulsando la **R**, deberá aparecer el menú colgante del que se elije de nuevo la opción **RUN**. Con esta serie de manipulaciones, el Turbo Pascal se encarga de **compilar** y **ejecutar** un programa fuente.

Cada Opción del menú de Turbo Pascal se selecciona con la tecla de la letra resaltada en la opción.

La salida del programa por pantalla:

```
UNIVERSIDAD
```

Notar que la respuesta es exactamente lo mismo que se encuentra en el programa entre comillas. Este es el argumento del procedimiento WRITE (escriba). Y es todo lo que hace el primer programa.

Otro ejemplo:

```
PROGRAM suma;
        VAR valor1, valor2, valor3, suma: INTEGER;
BEGIN
        WRITE('Entre un valor para primer sumando : ');
        READLN(valor1);
        WRITE('Entre un valor para segundo sumando : ');
        READLN(valor2);
        WRITE('Entre un valor para tercer sumando : ');
        READLN(valor3);
        suma:=valor1+valor2+valor3;
        WRITELN(valor1,' más ',valor2,' más ',valor3,' igual ',suma);
        READLN;
END.
```

La primera salida del programa por pantalla, debe ser:

```
Entre un valor para primer sumando : _
```

Hasta este momento se ha ejecutado la instrucción de la línea cuarta, pero la línea quinta contiene una instrucción de lectura y por lo tanto el programa espera hasta que se le introduzca un número entero cuyo valor se le va a asignar a la variable Valor1.
Después de digitar cualquier valor (por ejemplo 5) se debe pulsar <ENTER> y aparece:

```
Entre un valor para segundo sumando : _
```

Hasta este momento se ha ejecutado la instrucción de la línea sexta, pero la línea septima contiene una instrucción de lectura y por lo tanto el programa espera hasta que se le introduzca un número entero cuyo valor se le va a asignar a la variable Valor2.

Después de digitar cualquier valor (4) y pulsar <ENTER>, aparece:

```
Entre un valor para tercer sumando : _
```

Hasta este momento se ha ejecutado la instrucción de la línea octava, pero la línea novena contiene una instrucción de lectura y por lo tanto el programa espera hasta que se le introduzca un número entero

cuyo valor se le va a asignar a la variable Valor3. La línea 10 contiene una instrucción denominada de asignación, se interpreta en el sentido que a la variable suma se le asignará el resultado de sumar los tres valores (Valor1+Valor2+Valor3).

Después de digitar cualquier valor (2) pulsar <ENTER> aparece:

```
5 más 4 más 2 igual 11
```

y el programa ha terminado de correr.

Volver a correr presionando la letra R, e introduciendo otros valores.

Si se entran los números 20, 30 y 50, la salida del programa será:

```
20 más 30 más 50 igual 100
```

Modificar más el programa, de forma que quede:

```
PROGRAM suma;
        VAR valor1, valor2, valor3, suma, producto: INTEGER;
BEGIN
        WRITE('Entre un valor para primera variable  : ');
        READLN(valor1);
        WRITE('Entre un valor para segunda variable  : ');
        READLN(valor2);
        WRITE('Entre un valor para tercera variable  : ');
        READLN(valor3);
        suma:=valor1+valor2+valor3;
        producto:=valor1*valor2*valor3;
        WRITELN(valor1,'+',valor2,'+',valor3,'=',suma);
        WRITELN(valor1,'X',valor2,'X',valor3,'=',producto);
        READLN;
END.
```

Más ejemplos:

```
PROGRAM encuesta;
        VAR nombre, apellido1, apellido2: STRING[12];
BEGIN
        WRITE('Entre el nombre : ');
        READLN(nombre);
        WRITE('Entre el primer apellido : ');
        READLN(apellido1);
        WRITE('Entre el segundo apellido : ');
        READLN(apellido2);
        WRITELN('El nombre completo es: ');
        WRITE(nombre,'  ', apellido1,'  ',apellido2);
        READLN;
END.
```

Hay algo nuevo en la segunda línea del programa: la palabra reservada **STRING**; indica que las variables **nombre, apellido1 y apellido2** son de tipo cadena de caracteres, que pueden ser letras, signos especiales e inclusive números, pero con estos números no se pueden hacer operaciones de variables numéricas, [12] indica el número máximo de caracteres que se puede almacenar en estas variables. Correr este programa varias veces.

Modicar este programa y obtener este otro:

```
PROGRAM encuesta;
        VAR nombre, apellido1, apellido2: STRING[10];
        nota1,nota2,nota3,nota4,def: REAL;
BEGIN
        WRITE('Entre el nombre : ');
        READLN(nombre);
        WRITE('Entre el primer apellido : ');
        READLN(apellido1);
        WRITE('Entre el segundo apellido : ');
        READLN(apellido2);
        WRITE(apellido1,' por favor entra tu primera nota :');
        READLN(nota1);
        WRITE(nombre,' por favor entra tu segunda nota :');
        READLN(nota2);
        WRITE(nombre,' por favor entra tu tercera nota :');
        READLN(nota3);
        WRITE(nombre,' por favor entra tu cuarta nota :');
```

```
        READLN(nota4);
        def:=nota1*0.1+nota2*0.2+nota3*0.3+nota4*0.4;
        WRITE(apellido1,' ',apellido2,' ',nombre,' tu nota  es:');
        WRITE(def);
        READLN;
END.
```

En la tercera línea del programa: la palabra reservada REAL; indica que las variables **nota1, nota2, nota3, nota4 y def** son de tipo REAL, es decir son variables que manejan números con decimales a diferencia de los números enteros que no pueden manejar decimales. La línea de la sentencia de asignación **def:=nota1*0.1+nota2*0.2+nota3*0.3+nota4*0.4;** quiere decir que la operación para calcular la nota definitiva es tomar el 10% (es equivalente a multiplicar por 0.1) de la **nota1** más el 20% de la **nota2** (es equivalente a multiplicar por 0.2) más el 30% de la **nota3** y más el 40% de la **nota4**.

Correr este programa varias veces con diferentes valores tanto para las variables numéricas reales que almacenan las notas (REAL) como para las de cadenas de caracteres que almacenan los nombres y apellidos (STRING).

Modificar este programa de tal forma que sirva para calcular las notas definitivas de los estudiantes de la Universidad, sabiendo que sólo tienen tres notas; la primera nota vale el 30%, la segunda otro 30% y la tercera el 40%.

Hacer un programa que lea (READ o READLN) el nombre, la edad, el sexo, el teléfono y la dirección de una persona, el programa debe escribir (WRITE o WRITELN) por pantalla todos estos datos. Por ejemplo, dados los siguientes datos: Juan, 23, masculino, 865432, Cra. 4 # 5-45, el resultado del programa debe ser de la siguiente forma:

```
NOMBRE: Juan
EDAD: 23
SEXO: masculino
TELEFONO: 865432
DIRECCION: Cra. 4 # 5-45
```

## 2.7. Sentencia FOR

En todos los lenguajes se encuentran instrucciones de naturaleza cíclica. La sentencia FOR, realiza ciclos con una condición previamente definada, es decir, se conoce previamente cuántos ciclos o iteraciones se van a realizar.

En la sentencia **FOR** se utiliza una variable de tipo entero a la cual se le asigna inicialmente un valor, que se incrementa de uno para cada ciclo, aquí se ejecuta una sentencia o un conjunto de sentencias hasta que la variable entera toma cierto valor que se especifica en la misma sentencia.

Ejemplo:

```
PROGRAM numeros;
        VAR num: INTEGER;
BEGIN
        FOR num := 1 TO 10 DO
             WRITELN(num);
        READLN;
END.
```

La cuarta línea del programa tiene una instrucción **FOR** significa **para**, TO significa **a** o **hasta** y DO significa **haga**. Esta línea significa que **para (FOR)** la variable **num** se van a asignar (:=) los valores desde **1 hasta (TO) 10**, y en cada uno de estos casos se hace la escritura (WRITELN) de la variable **num**. Notar que al final de la línea cuarta no hay punto y coma (debe ser así), debido a que la instrucción **FOR** no ha terminado; en este caso la instrucción **WRITELN** está dentro de la instrucción **FOR**.

La salida del programa por pantalla:

```
1
2
3
4
5
6
7
8
9
10
```

Modificar el programa así:

```
PROGRAM numeros;
        VAR num: INTEGER;
BEGIN
        FOR num := 1 TO 20 DO
        WRITE(num);
        READLN;
END.
```

El cambio consiste en quitarle **ln** a la instrucción **WRITE** en la quinta línea, y en cambiar el 10 por 20 en la sexta línea.

Ahora la salida debe ser:

```
1234567891011121314151617181920
```

Cambiar ahora hasta tener el siguiente ejemplo:

```
PROGRAM tabla;
        VAR cando, cador, resul: INTEGER;
BEGIN
        WRITE(' Introduzca la tabla deseada :');
        READLN(cando);
        FOR cador := 1 TO 10 DO
        BEGIN
            resul:=cando*cador;
            WRITELN(cador,' X ',cando,'=',resul);
        END;
        READLN;
END.
```

La línea sexta ahora dice que la variable **cador** se le van a asignar los valores del 1 al 10, y mientras toma cada uno de estos valores se ejecutan dos instrucciones, por esto se necesita un bloque **BEGIN** (línea 7) y **END;** (línea 10), estas dos instrucciones que están dentro del bloque hacen parte de la instrucción **FOR**, decimos que el bloque está formado por estas dos instrucciones constituyen una sentencia compuesta. Para el caso, la necesidad es que cada vez que a **cador** se le asigne un valor, se realicen estas dos instrucciones, si no colocáramos este bloque, nuestro programa daría otra salida.

Al correr el programa, a la pregunta de la tabla deseada responder con el número 5, (y <ENTER> obviamente) el resultado de este programa es:

```
1 X 5 = 5
2 X 5 = 10
3 X 5 = 15
4 X 5 = 20
5 X 5 = 25
6 X 5 = 30
7 X 5 = 35
8 X 5 = 40
9 X 5 = 45
10 X 5 = 50
```

Correr varias veces (por lo menos 4 veces) este programa respondiendo diferentes valores.

De este programa cambiar la línea que dice: **WRITELN(cador,' X ',cando,'=',resul);** por: **WRITELN(cando,' X ',cador,'=',resul);**

Correr este programa por lo menos 4 veces. NOTAR LA DIFERENCIA.
Mirar la importancia de la sentencia compuesta en el ejemplo:

Hacer el siguiente cambio: suprimir **BEGIN** de la línea 7 y **END;** de la línea 10:

```
PROGRAM tabla;
        VAR cando, cador, resul: INTEGER;
BEGIN
        WRITE(' Introduzca la tabla deseada :');
        READLN(cando);
        FOR cador := 1 TO 10 DO
             resul:=cando*cador;
        WRITELN(cando,' X ',cador,'=',resul);
        READLN;
END.
```

Ahora modifíquese así:

```
PROGRAM tabla;
        VAR cando, cador: INTEGER;
BEGIN
        WRITE(' Introduzca la cantidad deseada : ');
        READLN(cando);
        FOR cador := 1 TO cando DO
            WRITE('R');
        READLN;
END.
```

Correr el programa respondiendo con el valor 9 y <ENTER>.
El resultado debe ser:

```
RRRRRRRRR
```

Correr el programa respondiendo con el valor 36 y <ENTER>.
El resultado debe ser:

```
RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
```

```
PROGRAM tabla;
        VAR cando, cador, num: INTEGER;
BEGIN
        WRITE(' Introduzca la cantidad deseada : ');
        READLN(num);
        FOR cador := 1 TO num DO
        BEGIN
            FOR cando := 1 TO num DO
                    WRITE('R');
            WRITELN;
        END;
        READLN;
END.
```

Se corre el programa y a la pregunta de la cantidad deseada se responde con el número 8, (y <ENTER> obviamente) el resultado de este programa es:

```
RRRRRRRR
RRRRRRRR
RRRRRRRR
RRRRRRRR
RRRRRRRR
RRRRRRRR
RRRRRRRR
RRRRRRRR
```

Nuestro nuevo programa tiene dos sentencias FOR; una dentro de otra. Notemos además que hay una sentencia **WRITELN** que no tiene parámetros; es decir, no tiene ni paréntesis, ni texto entre comillas, ni nombres de variables. Todo lo que hace es pasar el control de cursor a la siguiente línea cada vez que se cumple un ciclo de la primera sentencia FOR; es decir, pasa el cursor a la siguiente línea cada vez que haya escrito 8 erres.

Cambiar ahora en la instrucción **WRITE('R');** la **R** por **B**.

Correr varias veces el programa. Hacerle otros cambios, volver a correr y NOTAR SI HAY DIFERENCIAS.

En la sentencia **FOR cando := 1 TO num DO**
cambiar **num** por **cador**.

```
PROGRAM tabla;
        VAR cando, cador, num: INTEGER;
BEGIN
        WRITE(' Introduzca la cantidad deseada : ');
        READLN(num);
        FOR cador := 1 TO num DO
        BEGIN
            FOR cando := 1 TO cador DO
            WRITE('B');
            WRITELN;
        END;
        READLN;
END.
```

Corramos el programa y miremos el resultado, por ejemplo si respondemos a la pregunta con el número 10:

```
B
BB
BBB
BBBB
BBBBB
BBBBBB
BBBBBBB
BBBBBBBB
BBBBBBBBB
BBBBBBBBBB
```

Hacer un programa que lea (**READ o READLN**) el nombre, la edad, el sexo, el teléfono y la dirección de 8 personas, el programa debe escribir (**WRITE o WRITELN**) por pantalla todos estos datos. Ayuda: Al programa parecido de la sección anterior, introducir este proceso en una sentencia **FOR** con 8 ciclos. Notemos al correr el programa, que por cada persona se leen los datos y luego se escriben, que es diferente a leer los datos de las 8 personas y luego escribirlos (ejercicio que se verá más adelante).

Escribir un programa que genere sobre la pantalla las siguientes salidas:

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
1 2 3 4 5 6
1 2 3 4 5 6 7
```

Otro ejemplo: Se necesita hacer un programa que realice la sumatoria de los números naturales desde 1 hasta el número que se especifique, así, si se especifica el número 5, el resultado de la sumatoria sería: 1 + 2 + 3 + 4 + 5=15, si se especifica el número 7, el resultado de la sumatoria debe ser: 1 + 2 + 3 + 4 + 5 + 6 + 7=28

```
PROGRAM sumatoria;
VAR resultado, numero,i: INTEGER;
BEGIN
        WRITE('Escriba el numero ');
        READLN(numero);
        resultado:=0;
        FOR i:=1 TO numero DO
                resultado:=resultado+i;
        WRITELN('Sumatoria de primeros ',numero,' naturales es: ',resultado);
        READLN;

END.
```

## 2.8. Sentencia IF THEN ELSE

En la mayoría de los algoritmos se encuentran pasos que se realizan sí y sólo si hay dependencia funcional con el valor de una condición, es el caso en un problema donde de acuerdo al estado de una variable haya que realizar ciertas operaciones.

Un algoritmo sería:

1. **Si** voy en bus, **entonces**
2. Debo llevar el pasaje
3. **si no**
4. Debo salir temprano para ir caminando

```
PROGRAM chequeo;
        VAR numero, contador: INTEGER;
BEGIN
        WRITE(' Cual número de 1 a 20 no quiere que salga ? :');
        READLN(numero);
        FOR contador:= 1 TO 20 DO
            IF contador = numero THEN
                  WRITE(' no me gusta ')
            ELSE  WRITE(contador);
        READLN;
END.
```

En este programa, la línea séptima contiene la nueva instrucción: IF que significa **si** condicional (es una parte la instrucción), después de IF va una expresión que constituye una condición cuyo **valor será o cierto o falso**, en la línea octava hemos escrito otra parte de la instrucción con otra palabra especial THEN cuyo significado es **entonces** y luego de ésta palabra va una instrucción (para nuestro caso **WRITE('no me gusta')**) o un bloque de instrucciones si es necesario que se ejecuta únicamente si la condición es cierta, en la novena línea va otra palabra especial **ELSE** que significa **si no** y luego una instrucción (para nuestro caso **WRITE(contador);**) o un bloque de instrucciones (o incluso puede no existir), que se ejecuta si la condición es falsa.

La instrucción IF compara en cada ciclo de la instrucción **FOR** el valor de la variable **contador** con el valor de la variable **numero**; si es igual, es decir si **contador = numero** es **CIERTO entonces** (THEN) escribe **' no me gusta'** ; ahora, **si no** (ELSE) es así, es decir si **contador = numero** es FALSO escribe (**WRITE(contador)**) la variable **contador** (que va desde 1 hasta 20).

Analicemos un poco la sintaxis de esta instrucción: primero nos preguntaremos por qué después del **DO** en la instrucción **FOR** no va un bloque **BEGIN...END** ?, la respuesta es porque dentro del **FOR** se va a ejecutar una sentencia **IF THEN ELSE**, cuya estructura se explicó arriba; segundo por qué después de la séptima y de la octava líneas no va punto y coma ?, la respuesta es porque **IF... THEN... ELSE** es una instrucción.

La salida del programa por pantalla:

```
12345 no me gusta 78910111213141151617181920
```

Volver a correr el programa y responder con 10 a la pregunta, la salida debe ser:

```
123456789 no me gusta 111213141151617181920
```

Otro ejemplo:

```
PROGRAM adivinar;
USES CRT;
      VAR  pensado, adivina: INTEGER;
           nombre1, nombre2 : STRING[8];
BEGIN
      WRITELN('Entrar los nombres de los jugadores');
      WRITE(' Entre el nombre de quien piensa el número : ');
      READLN(nombre1);
      WRITE(' Entre el nombre de quien adivina el número : ');
      READLN(nombre2);
      WRITELN(nombre1,' Piensa un número entre 1 y 10');
      WRITE(nombre1, ' y dígamelo solo a mi : ');
      READLN(pensado);
      CLRSCR;
      WRITE(nombre2,' adivina el numero de ',nombre1,'  :');
      READLN(adivina);
      IF pensado=adivina THEN
           WRITELN(' Bravo ',nombre2)
      ELSE WRITELN(nombre2,' Fallaste, es : ',pensado);
      READLN;
END.
```

En nuestro programa hay una instrucción nueva: CLRSCR; lo que hace es limpiar la pantalla, notemos la importancia de esta instrucción en este caso: se necesita que quien va a adivinar el número no pueda verlo sobre pantalla. Para usar esta instrucción es necesario colocar en la cabecera del programa USES CRT.

Correr el programa varias veces.

Eliminar del programa la instrucción CLRSCR y notar la diferencia al correrlo.

Otras modificaciones:

```
PROGRAM adivinar;
USES CRT;
      VAR  pensado, adivina, inten : INTEGER;
           nombre1, nombre2 : STRING[15];
BEGIN
      WRITELN('Entremos los nombres de los jugadores');
      WRITE(' Entre el nombre de quien piensa el número : ');
      READLN(nombre1);
      WRITE(' Entre el nombre de quien adivina el número : ');
      READLN(nombre2);
      WRITELN(nombre1,' Piensa un número entre 1 y 10');
      WRITE(nombre1, ' y dígamelo solo a mí : ');
      READLN(pensado);
      CLRSCR;
      FOR inten:=1 TO 3 DO
      BEGIN
            WRITE(nombre2,' adivina el número de ',nombre1,' ? :');
            READLN(adivina);
            IF pensado=adivina THEN
                  WRITELN(' Bravo ',nombre2)
            ELSE WRITELN(nombre2,' Fallaste');
      END;
      READLN;
END.
```

Observar las correcciones: en la línea dos hay que declarar otra variable **inten** que va a controlar la sentencia **FOR** que hemos introducido en la línea después de **CLRSCR**. Nuestro programa le da tres intentos a la persona que adivina el número, esto se logra utilizando la sentencia **FOR**. Es importante que notemos, en este caso, que si se adivina el número en el primero o segundo intento, el programa correrá hasta el tercer intento (este problema lo solucionaremos más adelante con otra instrucción).

Modificar el programa anterior haciendo que el computador le dé pistas a quien adivina el número de la siguinte forma: si el número pensado es mayor, que coloque el mensaje 'es mayor'; si el número pensado es menor que coloque el mensaje 'es menor'. Ampliar más el rango de números a pensar (por ejemplo de 1 al 100). Si ha dado algo de dificultad mirar la siguiente ayuda:

```
BEGIN
     WRITE(nombre2,' adivina el numero de ',nombre1,' :');
     READLN(adivina);
     IF pensado=adivina THEN
          WRITELN(' Bravo ',nombre2)
     ELSE
          IF pensado > adivina THEN
               WRITELN(' es mayor')
          ELSE  WRITELN(' es menor');
END;
```

La segunda instrucción IF ( **IF pensado > adivina** ... ) está dentro del primer IF y es ejecutado solo si la primera condición (la del primer IF) es falsa, miremos cómo en esta instrucción IF la condición ha cambiado en el signo condicional; el signo (>) se lee y entiende como '**mayor que**'.

La siguiente tabla muestra otros signos que se pueden utilizar en las expresiones condicionales de acuerdo a nuestra necesidad:

| | |
|---|---|
| = | **igual;** |
| <> | **diferente;** |
| < | **menor que;** |
| <= | **menor o igual;** |
| > | **mayor que;** |
| >= | **mayor o igual;** |

Valor lógico de algunas expresiones:

2 = 2 es cierto (verdadero).
4+5 > 9 es falso.
45 < 6 es falso.
2+1 = 3 es cierto.
2 <> 2 es falso.
nu > p es cierto; dado que nu=12 y p=3.
3 <= 3 es cierto.
3 <= 4 es cierto.
4 >= 5 es falso.

Insertar una instrucción **CLRSCR** en el programa al inicio del bloque **BEGIN END** del **FOR**, (también **USES CRT** en la cabecera) y mirar la diferencia.

Hacer un programa que lea los nombres, las notas entre 0 y 10, de los cuatro períodos para 3 estudiantes, que calcule las notas definitivas y que muestre un reporte en donde diga nombre, nota definitiva y un mensaje de 'reprueba' o 'aprueba', según sea el caso así: si la nota definitiva es menor que 6 entonces reprueba, sino aprueba (nota es mayor o igual a 6).

Se necesita realizar un programa que calcule y muestre las raíces reales de un polinomio de la forma $P(x)=ax^2+bx+c$, si el polinomio tiene solución compleja, que muestre un mensaje relacionado con tal situación.

```
PROGRAM raices_polinomio;
VAR a,b,c,x1,x2,cansub:REAL;
BEGIN
        WRITE('Entre el valor del coeficiente de x² ');
        READLN(a);
        WRITE('Entre el valor del coeficiente de x ');
        READLN(b);
        WRITE('Entre el valor del término independiente ');
        READLN(c);
        cansub:=b*b-4*a*c;
        IF cansub < 0 THEN
                WRITELN('El polinomio tiene raíces complejas')
        ELSE
        BEGIN
                x1:=(-b+sqrt(cansub))/(2*a);
                x2;= (-b-sqrt(cansub))/(2*a);
                WRITELN('La primera raíz del polinomio es : ',x1:8:2);
                WRITELN('La segunda raíz del polinomio es : ',x2:8:2);
        END;
        READLN;
END.
```

## 2.9 Sentencia WHILE

Al igual que la sentencia FOR, ejecuta varias veces un conjunto de sentencias, pero no un número determinado de veces, la ejecuta mientras la condición sea verdadera.

Por ejemplo:

1. Mientras esté cansado.
2. Me quedaré en cama.

Otro ejemplo:

```
PROGRAM numeros;
        VAR ent: INTEGER;
BEGIN
        ent:=50;
        WHILE ent<60 DO
        BEGIN
             WRITE(ent);
             ent:=ent+1;
        END;
        READLN;
END.
```

La nueva instrucción se encuentra en la quinta línea del programa: **WHILE** significa **mientras** y **DO** significa **haga**. Esta línea se va a entender de la siguiente forma: mientras que la variable **ent** sea menor que 60, haga lo que se encuentra en el boque BEGIN .. END;. Dentro de este bloque hay una sentencia de asignación **ent:=ent+1**; esta sentencia se debe entender que a la variable denominada **ent** se le va a sumar el valor 1 (el nuevo valor de **ent** será el viejo valor que tenía más uno).

Se requiere hacer un programa que al introducir un número natural, aparezca un mensaje en pantalla que diga si es primo o no.

```
PROGRAM numero_primo;
VAR
        es_primo: BOOLEAN;
        numero,divisor: INTEGER;
BEGIN
        WRITE('Entre un número natural ');
        READLN(numero);
        es_primo:=TRUE;
        divisor:=2;
        WHILE (divisor <= numero DIV 2) AND es_primo DO
        BEGIN
                IF numero MOD divisor = 0 THEN
                        es_primo:=FALSE;
                divisor:=divisor+1;
        END;
        WRITE(numero);
        IF es_primo THEN
                WRITELN('  Es primo')
        ELSE
                WRITELN(' No es primo');
        READLN;
END.
```

## 2.10 Sentencia REPEAT ... UNTIL

Esta sentencia de ciclo o de iteración, ejecuta una o un conjunto de instrucciones al menos una vez, en los caso anteriores de instrucciones de ciclo (FOR y WHILE) si la condición no se cumple la primera vez que llega a la sentencia, las instrucciones no se ejecutan; en la sentencia REPEAT, se ejecuta o ejecutan las instrucciones y al final del primer ciclo se revisa la condición (UNTIL).

(Ver ejemplo del cajero Pág., 36).

## 2.11. Procedimientos

Un procedimiento es un fragmento de código, o sentencias, a las cuales se les puede asignar un nombre y se hace una declaración que convierte el grupo de sentencias en una unidad llamada procedimiento.

Se usará la metodología de diseño de programas de refinamiento por pasos, también conocida como de descomposición funcional, es una de las técnicas de programación estructurada.

Normalmente, un programa lo ven muchas personas además del programador original; y lo pueden llegar a modificar. Los procedimientos puede hacer más fácil esta labor de mantenimiento de sistemas de información, a la vez que reducen los costos.

Los procedimientos se usan principalmente para evitar la duplicidad de código, y conseguir programas más cortos, son también una herramienta conceptual para dividir un problema en subproblemas que se resuelven independientemente y facilitar la documentación los cuales pueden ser leídos y comprendidos independientemente.

Se debe ser capaz de resolver un problema ignorando lo que ocurre a su alrededor. Esta autodisciplina mental es necesaria para escribir programas de cualquier tamaño, y en especial los de gran volumen. Los procedimientos ofrecen un formalismo con el cual puede expresarse ese tipo de idea.

```
PROGRAM cajero;
USES crt;
VAR
        saldo, ingreso,retiro, opcion: INTEGER;
        clave: CHAR;
PROCEDURE menu;
BEGIN
        CLRSCR;
        WRITELN('MENU PRINCIPAL DEL CAJERO');
        WRITELN;
        WRITELN('1. Saldo');
        WRITELN('2. Retiro');
        WRITELN('3. Consigna');
        WRITELN('4. TERMINA');
        WRITELN('              Entre su opcion: ');
END;
PROCEDURE seleccion;
BEGIN
        IF opcion=1 THEN WRITE(saldo);
        IF opcion=2 THEN
        BEGIN
                WRITE('Entre el valor del retiro '); READLN(retiro);
                saldo:=saldo-retiro;
        END;
        IF opcion=3 THEN
        BEGIN
                WRITE('Entre el valor del ingreso '); READLN(ingreso);
                saldo:=saldo+ingreso;
        END;
END;
BEGIN
        Saldo:=0;    {El saldo inicial es para este ejemplo 0}
        WRITE('Inserte su tarjeta y digite su clave '); READLN(clave);
        IF clave = '5432' THEN
        REPEAT
                menu;
                READ(opcion);
                seleccion;
        UNTIL opcion=4;
        READLN;
END.
```

Otro ejemplo:

```
PROGRAM choza;
        PROCEDURE techo;
        BEGIN
                WRITELN('       C');
                WRITELN('    C    C');
                WRITELN('  C        C');
                WRITELN(' CCCCCC');
        END;
        PROCEDURE pared;
        BEGIN
                WRITELN(' CCCCCC');
                WRITELN(' C          C');
                WRITELN(' C          C');
                WRITELN(' CCCCCC');
        END;
BEGIN
        techo;
        pared;
        READLN;
END.
```

Este programa al correr hará una choza con la letra C, es importante mirar la nueva estructura del programa.

Observar que en la cabecera del programa hay dos bloques de instrucciones que empiezan con la palabra **PROCEDURE** cuyo significado es Procedimiento; un procedimiento podemos decir que es como un programa, que se puede utilizar dentro de otro programa. En nuestro caso el programa se llama **choza**, y tenemos definidos dos procedimientos: uno llamado **techo** y otro **pared**, estos procedimientos deben ir siempre en la cabecera del programa. Ahora, en el último bloque del programa (últimas cuatro líneas), se encuentra lo que se denomina, para este caso, el programa principal; éste contiene dos instrucciones que son exactamente los nombres de los procedimientos arriba definidos. Por lo tanto este programa hará: un **techo** y una **pared**; es decir una **choza**.

Correr el programa.

Ahora hacer esta modificacion al programa principal:

```
BEGIN
        pared;
        techo;
        READLN;
END.
```

Correr de nuevo el programa, y observar que el programa hace primero una pared y luego un techo.

Modicar más el bloque principal:

```
BEGIN
        pared;
        techo;
        pared;
        techo;
        READLN;
END.
```

Los nombres dados a los procedimientos, son ahora como nuevas instrucciones que nosotros mismos hemos definido en Turbo Pascal, y podríamos utilizarlas cuantas veces queramos en nuestro programa.

## 2.12. Parámetros

El concepto parámetro es común en la vida diaria. Por ejemplo, un cajero sabe cómo dar el cambio al adquirir algo. Para ello el cajero necesita conocer el costo del artículo, y cuando el cliente ha pagado, él hace el cálculo y le devuelve el cambio.

El parámetro solo puede utilizarse dentro del procedimiento o función en el que ha sido declarado. En ocasiones es necesario enviarle al procedimiento unos valores que previamente han sido definidos (Argumentos). Estos valores le permiten ejecutar al procedimiento una instrucción haciendo una correspondencia uno a uno entre el parámetro definido en el procedimiento y el argumento que lleva el llamado al mismo.

Ejemplo:

```
PROGRAM cajero;
USES crt;
VAR
        saldo, ingreso,retiro, opcion: INTEGER;   clave: CHAR;
PROCEDURE menu;
BEGIN
        CLRSCR;
        WRITELN('  Menú principal del Cajero');
        WRITELN('1. Saldo');
        WRITELN('2. Retiro');
        WRITELN('3. Consigna');
        WRITELN('4. Terminar');
        WRITELN('                    Entre la opción : ');
END;
PROCEDURE seleccion(opc:INTEGER);
BEGIN
        IF opc=1 THEN WRITE(saldo);
        IF opc=2 THEN
        BEGIN
                WRITE('Entre el valor del retiro ');
                READLN(retiro);
                saldo:=saldo-retiro;
        END;
        IF opc=3 THEN
        BEGIN
                WRITE('Entre el valor del ingreso '); READLN(ingreso);
                saldo:=saldo+ingreso;
        END;
END;
BEGIN
        Saldo:=0;    {El saldo inicial es para este ejemplo 0}
        WRITE('Inserte su tarjeta y digite su clave ');
        READLN(clave);
        IF clave = '5432' THEN
        REPEAT
                menu;
                READ(opcion);
                seleccion(opcion);
        UNTIL opcion=4;  READLN;
END.
```

Los parámetros en la sentencia de llamada se denominan parámetros actuales (argumentos), y los parámetros definidos en el procedimiento se denominan formales. A su vez, los parámetros pueden ser :

Parámetros por nombre o por valor: Se envía el valor de la variables (argumentos) y aunque en el ambiente del procedimiento el parámetro se altere cuando retorne quedará con el mismo valor la variable argumento con el que fue enviado. En el ejemplo anterior del cajero, el procedimiento **seleccion** utiliza un parámetro por valor llamado **opc** .

Parámetros por referencia: Si alteramos el valor almacenado en el procedimiento, al retorno ( o salir de este), el valor del argumento también cambia. La diferencia en al sintáxis entre la declaración del parámetro por valor y el parámetro por referencia es que éste lleva antecedida la palabra VAR.

Ejemplo:

```
PROGRAM resta;
        VAR minu,sust,dife: INTEGER;
PROCEDURE restar(VAR d:INTEGER; m,s:INTEGER);
BEGIN
        WRITE('Entre el minuendo ');
        READLN(m);
        WRITE('Entre el sustraendo ');
        READLN(s);
        d:=m-s;
END;
BEGIN
        restar(dife,minu,sust);
        WRITELN(minu,'-',sust,'=',dife);
        READLN;
END.
```

En el anterior programa, el parámetro **d** dentro del procedimiento **restar** está declarado por referencia, esto debe ser así por que de lo contrario la variable **dife** no sería modificada en su valor después de haber sido argumento al llamado del procedimiento. Modifiquemos el anterior programa de la siguiente forma para entender esto:

```
PROGRAM resta;
        VAR minu,sust,dife: INTEGER;
PROCEDURE restar(d:INTEGER; m,s:INTEGER);
BEGIN
        WRITE('Entre el minuendo ');
        READLN(m);
        WRITE('Entre el sustraendo ');
        READLN(s);
        d:=m-s;
END;
BEGIN
        dife:=345;
        restar(dife,minu,sust);
        WRITELN(minu,'-',sust,'=',dife);
        READLN;
END.
```

Al correr el programa podemos observar que el resultado sobre la variable **dife** siempre será **345**, debido a que el parámetro **d** se encuentra declarado por **valor**.

## 2.13. Funciones

Una función se utiliza para calcular un valor a partir de unos valores dados. Es un concepto análogo al de función matemática. Una función en Turbo Pascal es una relación uno a uno.

En los lenguajes de programación existen funciones que ya han sido definidas, ellas se encuentran en archivos que se llaman librerías, tal es el caso de la raíz cuadrada, valor absoluto, etc. Otras funciones son definidas por el usuario.

La diferencia entre un procedimiento y una función, es que aunque ambas son subconjuntos de instrucciones, las funciones al salir devuelven valores única y exclusivamente, una función puede ser utilizada como parte de una expresión en la instrucción.

El ejemplo anterior de la resta sería más precisamente definido como una función así:

```
PROGRAM resta;
        VAR minu,sust,dife: INTEGER;
PROCEDURE restar(m,s:INTEGER):INTEGER;
BEGIN
        restar:=m-s;
END;
BEGIN
        d:=345;
        WRITE('Entre el minuendo ');
        READLN(minu);
        WRITE('Entre el sustraendo ');
        READLN(sust);
        dife:=restar(minu,sust);
        WRITELN(minu,'-',sust,'=',dife);
        READLN;
END.
```

Volvamos al ejemplo de la sumatoria. Este procedimiento lo podemos convertir en función poniéndolo a retornar el valor de la suma.

```
PROGRAM sumatorias;
VAR numero1,total,i: INTEGER;
FUNCTION sumatoria(numero:INTEGER): INTEGER;
VAR resultado;
BEGIN
        resultado:=0;
        FOR i:=1 TO numero DO
                resultado:=resultado+i;
        sumatoria:=resultado;
END;
BEGIN
        WRITE('Escriba el numero ');
        READLN(numero1);
        total:=sumatoria(numero1);
        WRITELN('La sumatoria de los primeros ',numero1,' es: ',total);
        READLN;
END.
```

## 2.14. Variables globales y locales

Si tenemos una o más definiciones, ya sea de funciones o procedimientos, es posible declarar variables en cada procedimiento además de hacerlo en el programa principal, a esas variables se les denomina **locales**. Las variables **locales** son aquellas que sólo pertenecen al ambiente en el que se está trabajando, ya sea un procedimiento o una función. Es decir, si definimos en un procedimiento una variable, ésta es local al procedimiento y no puede ser usada por ningún otro procedimiento.

Las variables **globales** son aquellas variables que se definen en el programa principal y pueden ser usadas en cualquier procedimiento del programa.

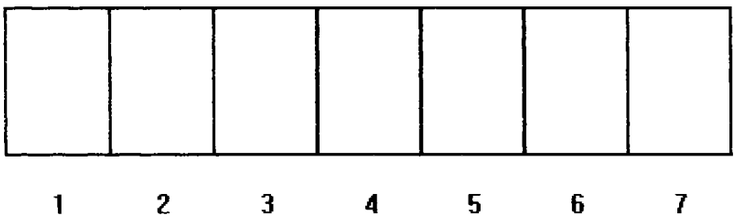
Observar al ejemplo anterior de la sumatoria, donde **i** es una variable declarada a nivel global y puede utilizarse sin ningún problema en función. La variable **resultado** ha sido declarada dentro de la función por lo tanto es una variable **local**; y sólo puede ser utilizada, o llamada dentro de la función, en el ambiente **global** no se reconoce.

Aunque la variable **numero1** es declarada como **global**, el valor es enviado como argumento a la función, la función no lo puede alterar debido a que es un parámetro por **valor**.

## 2.15 Arreglos

Existe, en la mayoría de los lenguajes de programación, la forma de crear tipos de datos más complejos que permiten hacer operaciones en forma más eficiente. Uno de estos tipos es el llamado **arreglo.** Un arreglo es una estructura tal que permite almacenar varios valores del mismo tipo. Podemos referirnos a un arreglo como un conjunto, o a cualquiera de sus elementos. Un ejemplo de arreglo son las cadenas de caracteres. Un vector es un caso de arreglo.

La forma más simple de un arreglo es el arreglo unidimensional, que puede ser definido abstractamente como un conjunto ordenado, finito de elementos homogéneos (del mismo tipo, o de iguales características). Como se puede ver en la siguiente figura.




Por "finito" se quiere decir que existe un número específico de elementos en el arreglo. Por "ordenado' se quiere decir que los elementos del arreglo están organizados de tal manera que existe un primero, segundo, tercero, etc.

Sin embargo, al especificar la estructura de datos, ésta no se describe completamente; debemos especificar cómo se puede almacenar o extraer datos de esa estructura. Un arreglo tiene dos tipos de datos asociados a él: El primer tipo, denominado **tipo base** del arreglo, es el tipo de elementos o componentes del arreglo. El Segundo tipo llamado **tipo índice** del arreglo es el tipo de los valores utilizados para ordenar los elementos individuales del arreglo.

Las dos operaciones básicas que se efectúan sobre un arreglo son extracción y almacenamiento. La operación de extracción es una función que acepta un arreglo a, y un elemento de su tipo de índice **i** y regresa un elemento del tipo base del arreglo. En Turbo Pascal, el resultado de la acción se representa por la declaración x:=a[i]. La operación de almacenamiento acepta un arreglo a, un elemento de su tipo de índice **i**, y un elemento de su tipo base **x**. Esta operación se representa por la declaración de asignación **a[i]:=x**. La operación se define mediante la regla: después de haber ejecutado la sentencia de asignación, el valor de **a[i]** es **x**. Antes de que se haya asignado el valor a un arreglo, su valor no esta definido y el referirse a él en una expresión es ilegal.

El tipo índice de un arreglo debe ser un tipo escalar o tipo subrango y el tipo base puede ser cualquier tipo válido en Turbo Pascal. El tamaño de un arreglo no puede cambiar durante la ejecución del programa. Existe un tipo especial de arreglos llamado arreglo empaquetado que puede ser utilizado para ahorrar espacio. Los detalles de cómo puede utilizarse este arreglo los puede buscar en un libro como: Estructura de Datos en Pascal de Tanembaum.

El siguiente ejemplo, muestra un arreglo de 100 elementos, donde cada uno de ellos es un número entero. Cada elemento se lee del teclado como un número entero, el cual es almacenado en la variable cuyo nombre es **a**,. Luego muestra uno a uno los valores de todos los cien números leídos y que se encuentran almacenados en el arreglo.

```
PROGRAM arreglo;
USES CRT;
        VAR a: ARRAY[1..100] OF INTEGER;
                i:INTEGER;
BEGIN
        WRITE('Digite los valores de  los elementos del vector ');
        FOR i:=1 TO 100 DO
        BEGIN
                WRITE('Entre el valor del elemento ',i,' = ');
                READLN(a[i]);
        END;
        CLRSCR;
        WRITELN('Estos son los valores del vector');
        FOR i:=1 TO 100 DO
        BEGIN
                WRITE(a[i]:8);
        END;
        READLN;
END.
```

### 2.15.1. Arreglos de una dimensión

Un arreglo unidimensional puede ser implementado fácilmente, consideremos primero arreglos cuyos tipos de índices son subrangos de enteros.

En Turbo Pascal todos los elementos de un arreglo tienen el mismo tamaño y tipo fijado con anterioridad, a su vez su dimensión debe ser definida previamente y no

puede variar durante el programa. Sin embargo, en algunos lenguajes de programación se permiten arreglos de objetos de cuyas dimensiones pueden variar durante el programa.

Para representar vectores se necesitan arreglos unidimensionales. Un vector es un arreglo unidimensional, cuyo tamaño se puede conocer usando lo siguiente fórmula.
Tamaño = limite superior - limite inferior + 1
Siendo el limite inferior el subíndice más pequeño del arreglo. El límite superior es el subíndice más grande que este puede alcanzar.

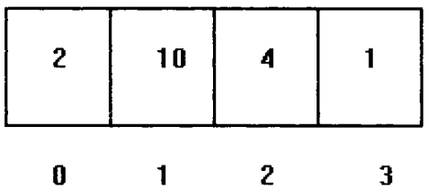
Al igual que el concepto de vector en matemáticas y en física, este puede aplicarse a estructuras de datos. El subíndice puede ser considerado el valor en el eje x, y el valor de la función en el eje de las y, es el elemento almacenado.

Ejemplo:

Representar un polinomio. El índice indicaría la potencia de la variable y el elemento, el coeficiente asociado a la potencia.

$y = x^3 + 4x^2 + 10x + 2$

Representación de vector:

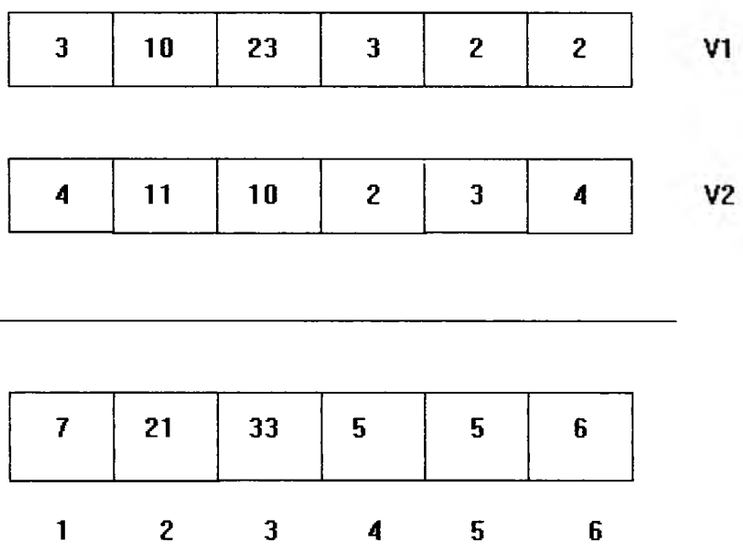



Con los vectores se pueden realizar varias operaciones:

A. Suma:

Para sumar dos vectores se debe tener en cuenta lo siguiente:

1. Los vectores deben ser de igual tamaño y tipo.
2. Se suman los elementos correspondientes al mismo subíndice.



```
PROGRAM sumavectores;
      USES CRT;
        TYPE vector= ARRAY[1..10] OF INTEGER;
        VAR a,b,c: vector;
                i, t:INTEGER;
PROCEDURE leevec(var v:vector; tamano:INTEGER; nombre: CHAR);
BEGIN
        WRITELN('Digite los valores de  los elementos del vector ',nombre);
        FOR i:=1 TO tamano DO
        BEGIN
                WRITE('Entre el valor del elemento ',i,'  = ');
                READLN(v[i]);
        END;
        WRITELN('Terminado de leer el vector ',nombre);
END;
PROCEDURE sumvec(VAR v3:vector; v1,v2:vector; tamano:INTEGER);
BEGIN
        FOR i:=1 TO tamano DO
         v3[i]:=v1[i]+v2[i];
END;
PROCEDURE escvec(v:vector; tamano:INTEGER; nombre:CHAR);
BEGIN
   WRITELN('Este es el vector ',nombre, ' :');
   FOR i:=1 TO tamano DO
      WRITE(v[i]:8);
   WRITELN;
END;
BEGIN
   CLRSCR;
   WRITE('Entre el tamaño del vector ');
   READLN(t);
   leevec(a,t,'a');
   leevec(b,t,'b');
   sumvec(c,a,b,t);
   escvec(a,t,'a');
   escvec(b,t,'b');
   WRITELN('Este es el vector que resulta de a+b');
   escvec(c,t,'c');
   READLN;
END.
```

La salida por pantalla será:

```
Entre el tamaño del vector  6
Digite los valores de  los elementos del vector a:
Entre el valor del elemento 1  = 3
Entre el valor del elemento 2  = 10
Entre el valor del elemento 3  = 23
Entre el valor del elemento 4  = 3
Entre el valor del elemento 5  = 2
Entre el valor del elemento 6  = 2
Terminado de leer el vector a
Digite los valores de  los elementos del vector b:
Entre el valor del elemento 1  = 4
Entre el valor del elemento 2  = 11
Entre el valor del elemento 3  = 10
Entre el valor del elemento 4  = 2
Entre el valor del elemento 5  = 3
Entre el valor del elemento 6  = 4
Terminado de leer el vector b
Este es el vector a:
     3     10     23      3      2      2
Este es el vector b:
     4     11     10      2      3      4
Este es el vector que resulta de a+b
Este es el vector c:
     7     21     33      5      5      6
```

B. Multiplicación:

1. El resultado es un escalar.

2. Los vectores deben ser de la misma longitud.

3. Se multiplica cada uno de los valores de los vectores correspondientes al mismo índice.

4. Luego se suma cada uno de los valores que resultan de la multiplicación.

VectorA: 3; 4; 5; 8;

VectorB: 1; 8; 2; 7;

El resultado sería: 3x1+4x8+5x2+8x7=101

```
PROGRAM produvectores;
      USES CRT;
        TYPE vector= ARRAY[1..10] OF INTEGER;
        VAR a,b: vector;
              i,result,t:INTEGER;
PROCEDURE leevec(var v:vector; tamano:INTEGER; nombre: CHAR);
BEGIN
        WRITELN('Digite los valores de  los elementos del vector ',nombre,'  : ');
        FOR i:=1 TO tamano DO
        BEGIN
              WRITE('Entre el valor del elemento ',i,'  = ');
              READLN(v[i]);
        END;
        WRITELN('Terminado de leer el vector ',nombre);
END;
FUNCTION mulvec(v1,v2:vector; tamano:INTEGER): INTEGER;
        VAR suma: INTEGER;
BEGIN
        suma:=0;
        FOR i:=1 TO tamano DO
              suma:=suma+v1[i]*v2[i];
        mulvec:=suma;
END;
PROCEDURE escvec(v:vector; tamano:INTEGER; nombre:CHAR);
BEGIN
   WRITELN('Este es el vector ',nombre,' :');
   FOR i:=1 TO tamano DO
      WRITE(v[i]:8);
   WRITELN;
END;
BEGIN
   CLRSCR;
   WRITE('Entre el tamaño del vector ');
   READLN(t);
   leevec(a,t,'a');
   leevec(b,t,'b');
   result:=mulvec(a,b,t);
   escvec(a,t,'a');
   escvec(b,t,'b');
   WRITELN('El producto de vectores a.b es ',result);
   READLN;
END.
```

El resultado por pantalla es:

```
Entre el tamaño del vector 4
Digite los valores de  los elementos del vector a:
Entre el valor del elemento 1  = 3
Entre el valor del elemento 2  = 4
Entre el valor del elemento 3  = 5
Entre el valor del elemento 4  = 8
Terminado de leer el vector a
Digite los valores de  los elementos del vector b:
Entre el valor del elemento 1  = 1
Entre el valor del elemento 2  = 8
Entre el valor del elemento 3  = 2
Entre el valor del elemento 4  = 7
Terminado de leer el vector b
Este es el vector a:
     3      4      5      8
Este es el vector b:
     1      8      2      7
El producto de vectores a.b es 101
```

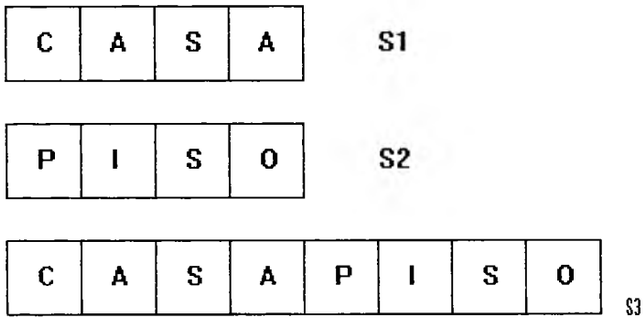
Una hilera, **STRING**, se define como arreglo de caracteres cuyo límite inferior es uno. Una hilera constante es denominada por medio de cualquier conjunto de caracteres incluidos entre comillas.

Los operadores de comparación pueden ser utilizados como parte de expresiones para operaciones entre arreglos de caracteres empacados. La comparación de caracteres se hace caracter por caracter hasta que se encuentre un caracter en la primera hilera que no es igual al correspondiente caracter de la segunda. Si tal caracter en la primera hilera está después del caracter de la segunda hilera, en el orden de caracteres en la máquina en particular que se ha usado, la primera hilera es mayor que la segunda; de otra manera la primera hilera es menor que la segunda. Si todos los caracteres son iguales, las dos hileras son iguales.

Otra característica de las hileras es que éstas pueden ser pasadas como argumentos en los procedimientos de salida.

Existe la operación de suma entre cadenas de caracteres:

**S3:=S1+S2;**




Hay funciones predefinidas en Turbo Pascal para el manejo de cadenas de caracteres, dos de ellas son: la función para encontrar la longitud de una hilera y la función encontrar una subcadena en una hilera. (Ver libro Joyanes Aguilar)

### 2.15.3. Arreglos de dos dimensiones

Un arreglo de dos dimensiones es una estructura de datos útil en programación y en la solución de problemas. Por ejemplo, se utiliza para describir un objeto que tiene físicamente dos dimensiones, tal es el caso de un mapa o un juego de damas. También para organizar un conjunto de valores que sean dependientes de dos valores de entrada. Por ejemplo un programa para un almacén de departamentos que tenga 20 surcusarles, cada una de las cuales vende 30 artículos, se puede tener un arreglo de 20 por 30.

```
PROGRAM arreglo;
    VAR a: ARRAY[1..10,1..10] OF INTEGER;
```

Un elemento de este arreglo puede ser manejado mediante la especificación de dos índices: número de fila y número de la columna.

Operaciones con matrices (arreglos de dos dimensiones)

A. Suma:

Para sumar dos matrices, éstas deben tener el mismo tamaño; el resultado será otra matriz del tamaño de las matrices sumadas. El elemento de cualquier posición para la matriz resultante será la suma de los elementos de esa misma posición en las matrices que se suman.

```
PROGRAM sumatrices;
USES CRT;
        TYPE matriz= ARRAY[1..10,1..10] OF REAL;
        VAR a,b,c: matriz; i,j,nfil,ncol:INTEGER;
PROCEDURE leemat(var ma:matriz; numfil,numcol:INTEGER; nom:CHAR);
BEGIN
        WRITELN('Digite los valores de  los elementos de la matriz ',nom, ':');
        FOR i:=1 TO numfil DO
        FOR j:=1 TO numcol DO
        BEGIN
                WRITE('Entre elemento fila',i,' columna ',j,' = ');
                READLN(ma[i,j]);
        END;   WRITELN('Termina de leer la matriz ',nom);
END;
PROCEDURE summat(VAR m1:matriz; m2,m3:matriz; nf,nc:INTEGER);
BEGIN
        FOR i:=1 TO nf DO
                FOR j:=1 TO nc DO
                        m1[i,j]:=m2[i,j]+m3[i,j];
END;
PROCEDURE escmat(v:matriz; f,c:INTEGER; nom:CHAR);
BEGIN
        WRITELN('Esta es la matriz ',nom,' :');
        FOR i:=1 TO f DO
        BEGIN
                FOR j:=1 TO c DO
                        WRITE(v[i,j]:8:2);
                WRITELN;
        END;
END;
BEGIN
        CLRSCR;
        WRITE('Entre el número de filas de las matrices a sumar ');
        READLN(nfil);
        WRITE('Entre el número de columnas de las matrices a sumar ');
        READLN(ncol);
        leemat(a,nfil,ncol,'a'); leemat(b,nfil,ncol,'b');
        summat(c,a,b,nfil,ncol);
        escmat(a,nfil,ncol,'a'); escmat(b,nfil,ncol,'b');
        WRITELN('Matriz resultante de la suma de a+b:');
        escmat(c,nfil,ncol,'c'); READLN;
END.
```

La salida por pantalla será:

```
Entre el número de filas de las matrices a sumar 2
Entre el número de columnas de las matrices a sumar 2
Digite los valores de  los elementos de la matriz a:
Entre elemento fila1 columna 1 = 4.3
Entre elemento fila1 columna 2 = 2.5
Entre elemento fila2 columna 1 = 6.4
Entre elemento fila2 columna 2 = 3.4
Termina de leer la matriz a:
Digite los valores de  los elementos de la matriz b:
Entre elemento fila1 columna 1 = 3.1
Entre elemento fila1 columna 2 = 2.5
Entre elemento fila2 columna 1 = 7.8
Entre elemento fila2 columna 2 = 6.9
Termina de leer la matriz b:
Esta es la matriz a :
   4.30   2.50
   6.40   3.40
Esta es la matriz b :
   3.10   2.50
   7.80   6.90
Matriz resultante de la suma de a+b:
Esta es la matriz c :
   7.40   5.00
  14.20  10.30
```

B. Multiplicación

En el producto de matrices, se deben cumplir los siguientes requisitos: como no existe la propiedad conmutativa, hay que diferenciar claramente las matrices que son efecto de la multiplicación: matriz que premultiplica y matriz que postmultiplica; el número de columnas de la matriz que premultiplica debe ser igual al número de filas de la matriz que postmultiplica y la matriz resultante del producto tendrá un tamaño igual por filas al número de filas de la matriz que premultiplica, y por columnas al número de columnas de la matriz que postmultiplica. Ejemplo : se multiplica A.B, si el tamaño de A es 3 filas y 2 columnas, el tamaño de la matriz B debe ser 2 filas y cualquier otro número determinado de columnas, puede ser el caso de 4; ahora, la matriz resultante C deberá tener el tamaño de 3 filas y 4 columnas.

Para obtener el valor de cada elemento de la matriz resultante en el producto, se debe hacer una sumatoria de todos los productos elemento a elemento de las filas en la matriz que premultiplica, con las columnas de la matriz que postmultiplica. Ejemplo:

Esta es la matriz a :

4.00 3.00

6.00 5.00

Esta es la matriz b :

6.00 7.00

5.00 4.00

Matriz resultante del producto a.b:

(4*6+3*5=39) (4*7+3*4=40)

(6*6+5*5=61) (6*7+5*4=62)

Quedando la matriz c:

39.00 40.00

61.00 62.00

```
PROGRAM mulmatrices;
USES CRT;
        TYPE matriz= ARRAY[1..10,1..10] OF REAL;
        VAR a,b,c: matriz; i,j,k,nfil,ncol,ncol2:INTEGER;

PROCEDURE leemat(var ma:matriz; numfil,numcol:INTEGER; nom:CHAR);
BEGIN
        WRITELN('Digite los valores de  los elementos de la matriz ',nom, ':');
        FOR i:=1 TO numfil DO
        FOR j:=1 TO numcol DO
        BEGIN
                WRITE('Entre elemento fila',i,' columna ',j,' = ');
                READLN(ma[i,j]);
        END;
        WRITELN('Termina de leer la matriz ',nom);
END;

PROCEDURE mulmat(VAR m1:matriz; m2,m3:matriz;
        nfa,nca,ncb:INTEGER);
BEGIN
        FOR i:=1 TO nfa DO
                FOR j:=1 TO ncb DO
                BEGIN
                        m1[i,j]:=0;
                        FOR k:=1 TO nca DO
                                m1[i,j]:=m1[i,j]+m2[i,k]*m3[k,j];
                END;
END;

PROCEDURE escmat(v:matriz; f,c:INTEGER; nom:CHAR);
BEGIN
        WRITELN('Esta es la matriz ',nom,' :');
        FOR i:=1 TO f DO
        BEGIN
                FOR j:=1 TO c DO
                        WRITE(v[i,j]:8:2);
                WRITELN;
        END;
END;
```

```
BEGIN
        CLRSCR;
        WRITE('Entre cuántas filas de tiene la 1a. matriz a multiplicar ');
        READLN(nfil);
        WRITE('Entre cuántas columnas tiene la 1a. matriz a multiplicar ');
        READLN(ncol);
        WRITE('Entre cuántas columnas tiene la 2a. matriz a multiplicar ');
        READLN(ncol2);
        leemat(a,nfil,ncol,'a'); leemat(b,ncol,ncol2,'b');
        mulmat(c,a,b,nfil,ncol,ncol2);
        escmat(a,nfil,ncol,'a'); escmat(b,ncol,ncol2,'b');
        WRITELN('Matriz resultante del producto a.b:');
        escmat(c,nfil,ncol2,'c'); READLN;
END.
```

Salida por pantalla:

```
Entre cuántas columnas tiene la 1a. matriz a multiplicar 2

Entre cuántas columnas tiene la 2a. matriz a multiplicar 2

Digite los valores de  los elementos de la matriz a:

Entre elemento fila1 columna 1 = 4

Entre elemento fila1 columna 2 = 3

Entre elemento fila2 columna 1 = 6

Entre elemento fila2 columna 2 = 5

Termina de leer la matriz a

Digite los valores de  los elementos de la matriz b:

Entre elemento fila1 columna 1 = 6

Entre elemento fila1 columna 2 = 7

Entre elemento fila2 columna 1 = 5

Entre elemento fila2 columna 2 = 4

Termina de leer la matriz b
```

```
Esta es la matriz a :
   4.00   3.00
   6.00   5.00
Esta es la matriz b :
   6.00   7.00
   5.00   4.00
Matriz resultante del producto a.b:
Esta es la matriz c :
  39.00  40.00
  61.00  62.00
```

### 2.15.4. Arreglos multidimensionales

Un ejemplo de arreglos tridimensionales los cuales se pueden declarar así:

```
VAR tri:ARRAY[1..5,1..12,1..15] OF INTEGER;
```

Un arreglo de este tipo está especificado por medio de tres subíndices tales como tri[i,j,k]. El primer índice especifica el número del plano, el segundo índice el número de la fila y el tercer el número de la columna. Este tipo de arreglo es útil cuando se determina un valor mediante tres entradas.

## 2.16. Conjuntos

Un conjunto es una colección de objetos. Todos los objetos de un conjunto deben ser del mismo **tipo base**. Este tipo base puede ser cualquier escalar o tipo subrango. Sin embargo, el número de valores que el tipo incluye puede ser severamente restringido por una implementación.

El conjunto que no tiene elementos es llamado conjunto vacío y se expresa como [].

La restricción en cuanto al número de elementos en el tipo base de un conjunto generalmente causa problemas al enumerar los elementos de un conjunto.

```
TYPE
        asignatura=(matematicas,calculo,geometria,informatica);
        pensum=SET OF asignatura;
VAR
        ingenieria,arquitectura: pensum;
```

### 2.16.1. Operaciones con conjuntos

Si **a** y **b** son conjuntos, su unión es un nuevo conjunto que contiene cada elemento que está en **a** o en **b**. La unión de dos conjuntos **a** y **b** se escribe como **a+b**.

La intersección de dos conjuntos **a** y **b** es un nuevo conjunto que tiene cada elemento que está tanto en **a** como en **b**. La intersección de **a** y **b** se escribe como **a*b**.

La diferencia de dos conjuntos, **a** y **b** es un nuevo conjunto que contiene cada elemento de **a** que no esté en **b**. La diferencia de **a** y **b** se escribe como **a-b**.

Los tipos bases de los conjuntos a los que se les aplican estas operaciones deben ser iguales o subrangos de éstos, es decir, deben ser compatibles.

Ejemplo:

```
PROGRAM menu;
USES CRT;
TYPE familia=(abuelo,abuela,tio,tia,padre,hermana,hermano,
        yo,esposa,hijo,hija);
VAR homio,hoabuelo,hoainte,hodIF,hotota: set of familia;
BEGIN
    CLRSCR;
    homio:=[esposa,tio,hijo,hija,yo];
    hoabuelo:=[abuelo,abuela,tio,tia,yo];
    hoainte:=hoabuelo*homio;
    IF tio IN hoainte  THEN
      WRITELN(' Mi tio pertenece al hogar de mi abuelo y al mio');
    IF yo IN hoainte THEN
      WRITELN(' Pertenezco al hogar de mi abuelo y al mio');
    hodIF:=hoabuelo-homio;
    IF abuela IN hodIF THEN
      WRITELN(' Abuela no pertenece a mi hogar, si al de mi abuelo');
    hotota:=homio+hoabuelo;
    IF hotota=[abuelo,abuela,tio,tia,esposa,hijo,hija,yo] THEN
      WRITELN('  Todos somos una gran familia');
  READLN;
END.
```

La salida por pantalla:

```
Mi tio pertenece al hogar de mi abuelo y al mio
Pertenezco al hogar de mi abuelo y al mio
Abuela no pertenece a mi hogar, si al de mi abuelo
Todos somos una gran familia
```

En este programa todas las expresiones booleanas de las intrucciones IF son verdaderas.

## 2.17. Recursión

Recursión es una técnica de programación que define un objeto en términos de una forma simple del mismo. Por ejemplo, una bahía, se puede considerar que está formada por bahías más pequeñas, y éstas a su vez por bahías más pequeñas, y así hasta encontrar un pedazo de tierra que ya no es una saliente de la tierra al mar. Un árbol, una bahía y figuras llamadas fractales, son recursivas.

Otro ejemplo es la función factorial:

La definición de factorial en forma iterativa es: n! = 1 * 2 * 3 * 4 * 5 * 6 .....* n

La forma recursiva es: n! = n * (n-1)!

En forma de algoritmo la función factorial (n) sería:

1. Si n = 0
2. factorial = 1
3. Sino
4. Factorial=n*factorial(n-1)

```
PROGRAM factorial;
  VAR numero, tabla: INTEGER;
FUNCTION fac(num: INTEGER):INTEGER;
BEGIN
  IF num=0 THEN fac:=1
  ELSE fac:=num*fac(num-1);
END;
BEGIN
 FOR numero:=1 TO 7 DO
 BEGIN
  tabla:=fac(numero);
  writeln(' el factorial de ',numero,' es ',tabla);
 END;
 READLN;
END.
```

Los resultados al correr el programa, son:

```
el factorial de 1 es 1
 el factorial de 2 es 2
 el factorial de 3 es 6
 el factorial de 4 es 24
 el factorial de 5 es 120
 el factorial de 6 es 720
 el factorial de 7 es 5040
```

Es importante en la recursión conocer la condición de escape, en el caso anterior era si n=0, ya que esta es la estructura más pequeña en la que se puede descomponer el factorial.

La función de Fibonacci, es una secuencia de números de la siguiente manera: 1,1,2,3,5,8,13,.... Observar que el siguiente número a partir del tercero es la suma de los dos anteriores.

```
PROGRAM fib;
  VAR numero, fibo: INTEGER;
FUNCTION fibonaci(n: INTEGER):INTEGER;
BEGIN
  IF n<=1 THEN fibonaci:=1
  ELSE fibonaci:=fibonaci(n-2)+fibonaci(n-1);
END;
BEGIN
 FOR numero:=1 TO 22 DO
 BEGIN
   fibo:=fibonaci(numero);
   writeln(' el fibonaci de ',numero,' es ',fibo);
 END;
 READLN;
END.
```

La salida de este programa es:

```
el fibonaci de 1 es 1
 el fibonaci de 2 es 2
 el fibonaci de 3 es 3
 el fibonaci de 4 es 5
 el fibonaci de 5 es 8
 el fibonaci de 6 es 13
 el fibonaci de 7 es 21
 el fibonaci de 8 es 34
 el fibonaci de 9 es 55
 el fibonaci de 10 es 89
 el fibonaci de 11 es 144
 el fibonaci de 12 es 233
 el fibonaci de 13 es 377
 el fibonaci de 14 es 610
 el fibonaci de 15 es 987
 el fibonaci de 16 es 1597
 el fibonaci de 17 es 2584
 el fibonaci de 18 es 4181
 el fibonaci de 19 es 6765
 el fibonaci de 20 es 10946
 el fibonaci de 21 es 17711
 el fibonaci de 22 es 28657
```

## 2.18. DOCUMENTACIÓN - Comentarios

Los comentarios facilitan la lectura de un programa; existen varias razones por las que interesa que un programa se lea fácilmente. Para enseñarlo a otra persona, especialmente cuando se le solicita ayuda para el trabajo ya que puede usarlo o modIficarlo, y si más tarde nosotros mismos debemos hacer un cambio, deben haber suficientes comentarios que nos hagan entender cómo el programa realiza la tarea.

Dentro de la documentación podemos hablar de la precondición de todo programa y de toda función o procedimiento que se encuentren en el programa. La precondición es parte de la documentación y en ella se expresa cómo es el sistema antes de entrar a resolver un problema. La postcondición expresa las salidas del sistema.

Los comentarios en un programa se encierran entre llaves, el compilador hará caso omiso de lo que allí se encuentre.

Ejemplo:

```
PROGRAM CONSTANTES;
CONST
  {Al frente de cada constante se escriben las unidades}
  KV1 = 3.79E-9;      {adimensional}
  KV2 = 3.82E-9;      {adimensional}
  KV3 = 4.35E-9;      {adimensional}
  KV4 = 3.43E-9;      {adimensional}
  Pat = 77500;        {pascales}
  RHO = 1000;         {Kilogramo/metro c£bico}
  CTE1 = 1.4E-5;      {adimensional}
  CTE2 = 1.39E-5;     {adimensional}
  CTE3 = 1.48E-5;     {adimensional}
  CTE4 = 1.52E-5;     {adimensional}...
```

## BIBLIOGRAFÍA


DIJKSTRA, E.W. A Discipline of Programming. Prentice-Hall. New Jersey. 1976.

GRIES, D. The Science of Programming. Springer Verlag. New York. 1981.

JACKSON, M.A. Principles of Program Design. Academic Press, London, 1975.

JOYANES AGUILAR, Luis. TURBO PASCAL 6.0 A SU ALCANCE. McGraw-Hill. España 1993.

KELLER, ARTHUR M. Programación en Pascal. McGraw-Hill de Mexico, S.A. de C.V. 1983.

ORGANICK E. I, FORSTHE A. I, PLUMMER B. P. PROGRAMMING LANGUAGE STRUCTURES. Academic Press.

STEPHEN, K. O'Brien. Turbo Pascal Manual de Referencia. Borland-Osborne/ McGraw-Hill de México, S.A. de C.V. 1990.

TAMAYO A., Alonso, Algoritmos Estructurados, Universidad Nacional Sede Manizales, 1991.

TENEBAUM, Aaron M, AUGENSTEIN. MOSHE J. ESTRUCTURA DE DATOS EN PASCAL. Prectice Hall International.

TREMBLAY, Jean-Paul/BUNT R. / OPSETH L. Pascal Estructurado. McGraw-Hill de México, S.A. de C.V. 1984.

ULLMAN. FUNDAMENTAL CONCEPTS OF PROGRAMING SYSTEMS. ADDISONWESLEY PUBLISHING COMPAY.

WATTT,David A. PROGRAMING LANGUAGE CONCEPTS And PARADIGMS. Prectice Hall International.

WELSH, J., ELDER, J. Introduction to Pascal. Prentice Hall, Inc. Englewood ClIFfs, N.J. 1979.

WIRTH, Niklaus. Program Development by Stepwise Refinement. CACM 14(4) pp. 221-227, Abril 1971.

WIRTH, Niklaus. Algorithms + Data Structures = Programs. Prentice Hall, Inc. Englewood ClIFfs, N.J. 1976.

WOOD, Jones. Turbo Pascal. Version 3.0.

ZAKS, Ronal. Pascal Turbo Pascal. Editorial REI. Bogotá.